Rem关于类型与效应代数重构的论文 s

PATRYCJ A BALIK、SZYMON JĘDRAS 和 PIOTR POLESIUK，弗罗茨瓦夫大学，波兰 d

在他们1991年的论文《类型与效应的代数重构》中，Pierre Jouvelot 和 David Gifford 提出了一种基于效应代数结构的类型与效应重建算法。他们的工作被视为类型与效应系统发展历程中的一个里程碑，并启发了静态分析领域后续的大量研究。然而，与其催生的后续研究不同，原始算法考虑的是一种支持高秩多态的语言，而这一特性很难正确实现。在本文中，我们指出了他们处理这一特性的方法中与变量绑定有关的细微错误。我们重新审视了他们的类型系统和重建算法，并描述了发现的问题。

arXiv:2601.15455v1 [cs.PL] 21 Jan 2026

1 引言

1991年，Pierre Jouvelot 和 David Gifford 发表了开创性论文《类型与效应的代数重构》[24]。他们的工作奠定了静态分析领域许多后续成果的基础（[5, 15, 17, 19, 29, 30, 35–37], 仅举数例），在这些成果中，编译器内部使用效应信息，但不会直接向程序员公开。然而，Jouvelot 和 Gifford 所考虑的原始语言和算法，将其效应系统视为一种面向用户的完整机制，其中包含显式的效应多态 lambda 和高秩效应多态函数。因此，他们预见了一项直到很久以后才受到更广泛关注的特性，例如，在带处理器的代数效应实用型类型与效应系统的语境中 [2–4, 8, 10, 20, 26, 27, 33, 39, 40]。

众所周知，高秩多态与类型（及效应）重建的结合在理论上颇为棘手。为了使问题仍处于可判定的范围内，必须保留一定数量的类型标注 [7, 13, 14, 16, 31, 32]。但即便克服了这一障碍，要设计出适用于这种情形的算法仍面临挑战，其中包括如何正确处理无处不在的类型变量绑定符——这类处理向来极易出错。阐明此类算法性质的定理，有助于消除人们对其相对于作为规范的声明式类型系统的可靠性和完备性的疑虑。然而，纸笔证明或许并不足够，因为人工验证时很容易漏掉细微的变量作用域问题。机械化证明能带来更高的可信度，但证明助手等工具经过相当长时间才发展到足以在编程语言领域广泛应用的成熟程度。即使是经典的类型重建算法 W，也是在 1999 [12, 28]，中才首次得到形式化，并且需要相当繁琐地处理变量绑定。

因此，《类型与效应的代数重构》中的证明当时需要手工验证，也就不足为奇了。我们发现了一些与变量相关的微妙错误，这些错误会导致算法拒绝某些有效程序，并在某些情况下推断出错误的类型。[1] 本文将原始类型系统与重构算法联系起来，并介绍我们对上述问题的发现。

2 类型系统

本节简要回顾语法和类型系统。由于原始呈现采用类似 LISP 的记法，我们将其改为更现代的惯例。此外，我们稍微改变了

[1]这些问题涉及演算中的高秩多态和显式多态特性。后续基于该算法得出的结果仍然有效，因为这些结果并未采用这些机制。

作者联系信息：Patrycja Balik，pbalik@cs.uni.wroc.pl；Szymon Jędras，sjedras@cs.uni.wroc.pl；Piotr Polesiuk，ppolesiuk@cs.uni.wroc.pl，波兰弗罗茨瓦夫弗罗茨瓦夫大学计算机科学研究所。

α， β， γ，……（类型变量）?X，?Y，?Z……（统一变量）
I ::= α | ?X（标识符）x， y， z， f，……（项变量）κ ::=E | T
（种类）（效应）（类型）D ::= ε | τ （描述符）e ::= x | λx：
$\tau . e$ | $\lambda x . e$ | $e\ e$ | $\Lambda \alpha^{\kappa} . e$ | e [D]（表达式）

ε ::= α | ?X | ι | $\varepsilon \cdot \varepsilon$
τ ::= α | ?X | $\tau \to_{\varepsilon} \tau$ | $\forall \alpha^{\kappa} . \tau$

图1. 语法。

将变量绑定符表示为更卫生但仍等价的形式的方法。我们将在第2.2节中详细阐述。

2.1 语法

语法见图1。我们区分类型变量和统一变量，并引入标识符这一句法类别，标识符可以是其中任意一种。由于标识符既可以表示类型，也可以表示效应，因此我们用两种种类来区分它们：所有类型的种类T以及所有效应的种类E。效应由类型变量、统一变量、纯效应 ι以及两个效应的合并 $\varepsilon_1 \cdot \varepsilon_2$构成。类型包括类型变量、统一变量、带效应标注的箭头类型 $\tau_1 \to_{\varepsilon} \tau_2$和多态类型 $\forall \alpha^{\kappa} . \tau$，后者在类型体中绑定给定种类的类型变量。描述符可以是效应或类型。

表达式的语法包括项变量 x、带类型标注的 λ 抽象λx： $\tau . e$、不带类型标注的 λ 抽象 $\lambda x . e$、函数应用 $e_1\ e_2$、类型抽象 $\Lambda \alpha^{\kappa} . e$ 和类型应用 e [D]。直观而言，未标注 λ 抽象的参数类型需要推断，但有一项限制：它必须是单态的即，不能包含任何多态类型。形式上，单态类型由以下归纳关系定义。

$$\frac{}{\mathsf{mono}(\alpha)}\ \text{MT-Var} \qquad \frac{}{\mathsf{mono}(?\mathrm{X})}\ \text{MT-UnifVar} \qquad \frac{\mathsf{mono}(\tau_1) \qquad \mathsf{mono}(\tau_2)}{\mathsf{mono}(\tau_1 \to_{\varepsilon} \tau_2)}\ \text{MT-Arrow}$$

2.2 与原始呈现的差异

所呈现的语法与原始语法略有不同，但在我们看来，这些改动提升了可读性，且不会改变系统的本质。下面简要介绍这些差异。

变量。在原始呈现中，类型中的绑定符可以根据语法绑定标识符（类型变量或统一变量）。由于统一变量的绑定符绝不会出现在源程序或算法的输出中，我们决定在语法中禁止这种情况。原始呈现还将标识符这一句法类别用于项变量。由于项变量不会与类型变量或统一变量混用，我们也将其视为一个独立的句法类别。

标识符的原始语法允许使用常量。常量有两种用途：一是作为在程序外部定义的全局变量，二是作为算法生成的新鲜标识符，用于表示由多态量词引入的未知抽象类型。在我们的介绍中

$$\frac{(\alpha :: \kappa) \in \Gamma}{\Gamma \vdash \alpha :: \kappa}\ \text{K-Var} \qquad \frac{\Gamma \vdash \tau_1 :: \mathrm{T} \qquad \Gamma \vdash \tau_2 :: \mathrm{T} \qquad \Gamma \vdash \varepsilon :: \mathrm{E}}{\Gamma \vdash \tau_1 \to_\varepsilon \tau_2 :: \mathrm{T}}\ \text{K-Arrow}$$

$$\frac{\Gamma,\ \alpha :: \kappa \vdash \tau :: \mathrm{T}}{\Gamma \vdash \forall \alpha^\kappa.\, \tau :: \mathrm{T}}\ \text{K-Forall} \qquad \frac{}{\Gamma \vdash \iota :: \mathrm{E}}\ \text{K-纯效应} \qquad \frac{\Gamma \vdash \varepsilon_1 :: \mathrm{E} \qquad \Gamma \vdash \varepsilon_2 :: \mathrm{E}}{\Gamma \vdash \varepsilon_1 \cdot \varepsilon_2 :: \mathrm{E}}\ \text{K-合并}$$

图2。种类判断规则。

我们对这两种用途都使用常规的类型变量和项变量，从而无需单独的句法类别。

纯效应。 在原始呈现中，有一个专门表示纯效应的常量。由于我们将常量视为普通类型变量，因此决定将纯效应表示为效应语法中的一个独立构造 ι。

单态类型。 最初，单态类型由一个独立的句法类别定义，其中包括标识符和箭头类型。由于我们倾向于采用证明助手中的形式化方式，因此希望将独立的句法类别视为独立的归纳类型。因此，我们将单态类型定义为（多态）类型上的一元关系。

2.3 类型系统

我们从 图2 中所示的种类判断规则开始介绍类型系统。种类判断关系具有双重作用。首先，它确保类型和效应的形式良好，即所有类型变量和效应变量都在环境中有绑定。其次，由于该关系定义在描述符上，因此在需要时，它可以让我们区分类型和效应。

类型系统的下一个组成部分是类型等价（$\tau_1 \equiv \tau_2$）和效应等价（$\varepsilon_1 \equiv \varepsilon_2$）关系。效应等价关系定义为包含幂等交换幺半群各定律的最小同余关系，其中 ι 为单位元，$\cdot$ 为二元运算。在效应等价关系的基础上，我们将类型等价关系定义为满足箭头类型中的效应等价关系，并允许对多态类型中绑定的类型变量和效应变量进行 α-重命名的最小同余关系。由于这些定义都是标准的，我们在此省略。

最后，我们在图3中给出声明式类型系统的类型规则。大多数规则都是标准的，因此我们只着重说明几个细节。对于无注解λ抽象的规则T-LamU，我们要求参数类型是单态的，因为类型推断算法无法推断多态类型。在类型应用的规则T-AppD 中，我们检查描述符是否具有预期的种类，并使用避免捕获的替换将其代入多态类型的主体中（我们用符号 $\theta^* A$ 表示将替换 θ 应用于项 A）。T-Conv 规则允许将表达式的类型和效应改为与之等价的类型和效应。

值得注意的是，规则T-Lam 和 T-LamU 中要求参数类型满足良构性（以 蓝色标出）的前提在原始表述中被省略了。鉴于这类前提十分常见，我们认为这只是一个小疏忽，因此在本文的表述中将其补上。

3 合一

为了支持类型推断，Jouvelot 和 Gifford 设计了一种合一算法，它能根据上一节介绍的等价关系对类型进行合一。他们方法的新颖之处在于

$$\frac{(x:\tau)\in\Gamma}{\Gamma\vdash x:\tau\ /\ \iota}\ \textsc{T-Var} \qquad \frac{\Gamma\vdash\tau_1::\mathsf{T} \qquad \Gamma,x:\tau_1\vdash e:\tau_2\ /\ \varepsilon}{\Gamma\vdash\lambda x:\tau_1.e:\tau_1\to_\varepsilon\tau_2\ /\ \iota}\ \textsc{T-Lam}$$

$$\frac{\Gamma\vdash\tau_1::\mathsf{T} \qquad \mathsf{mono}(\tau_1) \qquad \Gamma,x:\tau_1\vdash e:\tau_2\ /\ \varepsilon}{\Gamma\vdash\lambda x.e:\tau_1\to_\varepsilon\tau_2\ /\ \iota}\ \textsc{T-LamU}$$

$$\frac{\Gamma\vdash e_1:\tau_2\to_\varepsilon\tau_1\ /\ \varepsilon_1 \qquad \Gamma\vdash e_2:\tau_2\ /\ \varepsilon_2}{\Gamma\vdash e_1\ e_2:\tau_1\ /\ \varepsilon_1\cdot\varepsilon_2\cdot\varepsilon}\ \textsc{T-App} \qquad \frac{\Gamma,\alpha::\kappa\vdash e:\tau\ /\ \iota}{\Gamma\vdash\Lambda\alpha^\kappa.e:\forall\alpha^\kappa.\tau\ /\ \iota}\ \textsc{T-LamD}$$

$$\frac{\Gamma\vdash e:\forall\alpha^\kappa.\tau\ /\ \varepsilon \qquad \Gamma\vdash\mathsf{D}::\kappa}{\Gamma\vdash e\ [\mathsf{D}]:\{\alpha\mapsto\mathsf{D}\}^*\tau\ /\ \varepsilon}\ \textsc{T-AppD} \qquad \frac{\Gamma\vdash e:\tau_1\ /\ \varepsilon_1 \qquad \tau_1\equiv\tau_2 \qquad \varepsilon_1\equiv\varepsilon_2}{\Gamma\vdash e:\tau_2\ /\ \varepsilon_2}\ \textsc{T-Conv}$$

图 3. 声明式类型系统的类型规则。

统一(?X; τ) =
  若 τ = ?X，则 $\langle\emptyset;\emptyset\rangle$
  若 ?X $\in$ UV(τ)，则失败
  若 单态(τ)，则 $\langle[?X\mapsto\tau];\emptyset\rangle$
  否则失败
统一(τ; ?X) = 统一(?X; τ)

统一($\tau_1\to_{\varepsilon_1}\tau_1'$; $\tau_2\to_{\varepsilon_2}\tau_2'$) =
  let $\langle\theta_1;\ \Omega_1\rangle$ = 统一($\tau_1;\tau_2$) **in**
  令 $\langle\theta_2;\ \Omega_2\rangle$ = 统一($\theta_1^*\tau_1'$; $\theta_1^*\tau_2'$)于
  $\theta_2\theta_1$; $\Omega_1\cup\Omega_2\cup\{\varepsilon_1\equiv\varepsilon_2\}$
统一($\forall\alpha_1^\kappa.\tau_1$; $\forall\alpha_2^\kappa.\tau_2$) =
  新鲜生成 β 于
  统一($[\alpha_1\mapsto\beta]^*\tau_1$; $[\alpha_2\mapsto\beta]^*\tau_2$)

图 4. 代数统一算法。

合一会产生一个替换，使类型结构相等，但会延后效应的合一，并生成一组需要稍后求解的等式约束。

我们在图 4中给出了他们的合一算法。该算法接受两个类型，并返回一个由两部分组成的结果：针对合一变量的替换，以及效应之间的约束集。该算法据称基于 Robinson统一算法 [34], ，但原始呈现中省略了“相同变量情形”和出现检查（两者均以蓝色标示）。我们认为这只是一个小疏忽，因此在本文的呈现中将其补上。

由于合一变量表示某些未知的单态类型，在第一种“变量与类型”情形中，算法会在生成替换之前额外检查该类型是否为单态类型。要合一两个箭头类型，算法会递归地合一它们的参数类型和结果类型，但会将效应的合一延后，通过向结果中添加一个约束来处理。最后一种情形处理多态类型的合一。由于两个类型可能使用不同的绑定类型变量，算法首先将两个绑定变量都重命名为同一个新鲜生成的变量 β，然后递归地合一多态类型的主体。如果以上情形都不匹配，算法就会失败。请注意，我们在算法中使用的替换记法与声明式类型系统中的不同。下一节将解释其中的区别。

3.1 合一变量的语义

要理解合一算法，我们必须明确类型量词之下出现的合一变量的预期语义，例如 $\tau \triangleq \forall \alpha^{\mathsf{T}}.?\mathrm{X} \to_{\iota} \alpha$ 这样的类型。可能有两种语义。

> **捕获语义**。 在捕获语义下，合一变量可以实例化为包含绑定类型变量的类型，从而导致变量捕获。在上面的示例中，合一变量 ?X 可以实例化为 α，从而使类型 τ实例化为 $\forall \alpha^{\mathsf{T}}.\, \alpha \to_{\iota} \alpha$。正确处理合一的捕获语义通常存在问题，但借助显式替换 [11, 21] 或名称置换 [38]可以实现。

> **避免捕获语义**。 在避免捕获语义下，类型变量不会因实例化合一变量而被捕获。使用标准的避免捕获的替换时，避免捕获语义更易于实现。另一方面，它的表达能力较弱，因此会危及类型推断算法的完备性。

原始呈现没有明确说明使用哪种语义。不过，有若干线索。首先，类型推断算法被声称是完备的，因此更具表达能力的捕获语义很可能才是预期语义。其次，正如我们将在 第 4 节中看到的，类型应用情形的类型推断算法会对一个未被绑定的类型变量进行替换，但（可能是有意地）使其泄漏出自己的作用域。这表明，绑定类型变量所使用的实际标识符很重要，而这与避免捕获替换并不一致，因为避免捕获替换可能会改变绑定变量的名称。第三，如果假定使用避免捕获替换，那么所给出的完备性证明部分将是不健全的。

原论文的一位作者在私人通信中确认：“他不记得曾禁止‘局部’合一[2] [23]。”此外，他表示，他们原本打算通过假设整个程序中的所有绑定变量都具有不同的名称，来解决变量捕获问题。论文自身也支持这一点，因为文中假设整个程序都经过“阿尔法重命名，因此不存在标识符名称冲突”。

总而言之，我们得出以下结论：

- 合一变量的预期语义是捕获语义，
- 通过对整个程序进行 α 重命名，确保所有绑定变量的名称各不相同，从而解决作用域问题，因此使用避免捕获的替换还是捕获替换并无影响，
- 类型推断算法要求保留绑定变量的名称，因此使用捕获替换更为精确，
- 完备性证明似乎依赖于捕获替换。

为了确保表述精确，并区分避免捕获的替换（声明式类型系统所用）和捕获替换（算法所用），我们用花括号表示避免捕获的替换（$\{\alpha \mapsto \tau\}$），用方括号表示捕获替换（$[\alpha \mapsto \tau]$）。

3.2 捕获替换

有些读者可能会对捕获替换感到不自在，因为这种做法并不常见（至少在编程语言元理论中如此），而且如果处理不慎，可能会导致意料之外的变量捕获。为了让这些读者相信，捕获替换并没有看起来那么糟糕，我们来考虑一种简化声明式类型系统，它禁止未标注的

[2]即，具有捕获语义的合一

λ抽象。就本小节而言，我们采用图3中的类型系统，但移除规则 T-LamU。

对于这种简化的类型系统，可以定义一种不使用合一的简单类型推断算法。我们来考察一种使用捕获替换的此类算法变体。算法中最有趣的两种情况如下。

s-infer($\Gamma; e_1\ e_2$) =
  令 $\tau_a \to_\varepsilon \tau_1;\ \varepsilon_1$ = s-infer($\Gamma; e_1$) **于**
  令 $\tau_2;\ \varepsilon_2$ = s-infer($\Gamma; e_2$) **于**
  若 $\tau_a \equiv \tau_2$ **则** $\langle \tau_1; \varepsilon_1 \cdot\ \varepsilon_2 \cdot\ \varepsilon \rangle$
  否则失败

s-推断($\Gamma; e\ [\mathrm{D}]$) =
  令 $\forall \alpha^\kappa.\ \tau;\ \varepsilon$ = s-infer($\Gamma;\ e$) **于**
  若 $\Gamma \vdash \mathrm{D} ::\ \kappa$ **则** $\langle [\alpha \mapsto \mathrm{D}]^* \tau; \varepsilon \rangle$
  否则失败

在应用情形中，我们推断函数及其参数的类型，确保函数具有箭头类型，并检查参数类型是否与预期类型匹配。由于我们没有统一变量，只需检查类型的 α-等价性（模效应的等式理论）。在类型应用情况中，我们推断表达式的类型，确保它是一个多态类型，绑定了一个种类符合预期的类型变量，然后使用捕获替换，将描述符替换绑定的类型变量。

证明该算法相对于简化声明式类型系统是可靠且完备的过程相当技术性，因此我们在此仅概述主要思路。首先，我们定义以下两种绑定变量唯一性的概念。

定义 *3.1*。 对于任意句法类别的给定项 A （例如，表达式、类型、类型环境），如果对于环境 Γ，其中每个绑定类型变量或效应变量 α 在 A中仅绑定一次，且不属于环境 Γ 的定义域（即，对任意种类 κ，均不存在 $(\alpha :: \kappa) \notin \Gamma$），则称 A 相对于环境 Γ具有全局唯一绑定变量（记作 $\Gamma \vdash A$ G-unique）。如果每个绑定类型变量或效应变量 α 在 A 中都不会遮蔽 A中的任何其他绑定变量，且不属于环境 Γ的定义域，则称 A相对于环境 Γ 具有局部唯一绑定变量（记作 $\Gamma \vdash A$ L-unique）。特别地，若对于每个类型赋值 $(x : \tau) \in \Gamma$，都有 $\Gamma \vdash \tau$ L-unique，则类型环境 Γ 相对于其自身具有局部唯一绑定变量（$\Gamma \vdash \Gamma$ L-unique）。

示例 *3.2.* 考虑环境 $\Gamma = [\alpha :: \mathrm{T}\ x :\ \forall \beta^{\mathrm{T}}.\ \beta]$。

- 我们有 $\varnothing \vdash \forall \alpha^{\mathrm{T}}.\ \alpha \to_\iota \alpha$G-唯一，且 $\varnothing \vdash \forall \alpha^{\mathrm{T}}.\ \alpha \to_\iota \alpha$L-唯一，但 $\Gamma \vdash \forall \alpha^{\mathrm{T}}.\ \alpha \to_\iota \alpha$G-唯一和 $\Gamma \vdash \forall \alpha^{\mathrm{T}}.\ \alpha \to_\iota \alpha$L-唯一均不成立，因为 α位于 Γ的定义域中。
- 我们有 $\Gamma \vdash \forall \beta^{\mathrm{T}}.\ \beta$ G-唯一，且 $\Gamma \vdash \forall \beta^{\mathrm{T}}.\ \beta$ L-唯一，尽管 β 在 Γ和类型中都已绑定。
- 我们有 $\Gamma \vdash (\forall \beta^{\mathrm{T}}.\ \beta) \to_\iota (\forall \beta^{\mathrm{T}}.\ \beta)$ L-唯一，因为 β 的两个绑定不会相互遮蔽。另一方面，$\Gamma \vdash (\forall \beta^{\mathrm{T}}.\ \beta) \to_\iota (\forall \beta^{\mathrm{T}}.\ \beta)$ G-唯一不成立，因为β 被绑定了两次。
- 由于 γ 被绑定了两次，且第二次绑定遮蔽了第一次绑定，因此 $\Gamma \vdash \forall \gamma^{\mathrm{T}}.\ \gamma \to_\iota \forall \gamma^{\mathrm{T}}.\ \gamma$G-unique 和 $\Gamma \vdash \forall \gamma^{\mathrm{T}}.\ \gamma \to_\iota \forall \gamma^{\mathrm{T}}.\ \gamma$L-unique 均不成立。
- 我们有 $\Gamma \vdash \Gamma$ L-unique，因为我们有 $\Gamma \vdash \forall \beta^{\mathrm{T}}.\ \beta$ L-unique。

根据这些概念，我们可以陈述正确性引理：当满足某些唯一性条件时，捕获替换与避免捕获的替换一致。由于类型推断算法返回的类型不一定具有全局唯一绑定变量，因此该引理针对具有局部唯一绑定变量的类型进行陈述。

引理 3.3。 若 $\Gamma, \alpha :: \kappa \vdash \tau$ 局部唯一 且 $\Gamma \vdash \mathrm{D} :: \kappa$ 成立，则 $\{\alpha \mapsto \mathrm{D}\}^*\tau \equiv [\alpha \mapsto \mathrm{D}]^*\tau$。此外，若还满足 $\Gamma \vdash \mathrm{D}$ 局部唯一，则 $\Gamma \vdash \{\alpha \mapsto \mathrm{D}\}^*\tau$ 局部唯一 成立。

为确保在类型推断算法中使用替换时满足引理的要求，我们可以证明该算法满足以下不变式。

引理 3.4. 假设 s-infer$(\Gamma; e) = \langle \tau; \varepsilon \rangle$ 并且类型推断算法的输入满足以下条件：

- $\Gamma \vdash e$ G-唯一，
- $\Gamma \vdash \Gamma$ L-唯一，
- Γ 和 e 的绑定变量集合不相交。

那么， 以下条件成立：

- $\Gamma \vdash \tau$ L-唯一，
- τ 中的每个绑定变量都在 e 或 Γ中绑定。

有了这一不变式和 引理 3.3，类型推断算法的可靠性和完备性证明便变得直截了当。

3.3 批评

在原始呈现中，合一算法被认为是正确的。由于该算法不直接统一效应，而是生成它们之间的约束，因此正确性陈述使用了以下模型概念。

定义 *3.5*（模型）。如果捕获替换 m 将效应合一变量映射到基态效应，我们就称它为模型。如果模型 m 满足约束集 Ω（记作 $m \models \Omega$），那么对于每个约束 $\varepsilon_1 \equiv \varepsilon_2 \in \Omega$，都有 $m^*\varepsilon_1 \equiv m^*\varepsilon_2$。

我们不清楚这里的基态效应是什么意思。当然，基态效应不能包含合一变量，但它们能否包含常规效应变量或常量？不过，这一点对于我们发现的问题并不重要。

原始正确性定理表述如下。

定理 3.6（正确性， [24]中的定理 1）。如果$\mathsf{unify}(\tau_1; \tau_2) = \theta; \Omega$ 并且 $m \models \Omega$ 对于某个模型 m，那么 $m^*\theta^*\tau_1 \equiv m^*\theta^*\tau_2$。

文中没有给出证明细节，但据称证明可通过对类型的结构进行归纳得到。我们发现，合一算法在多态类型的情形下表现不正确。请看以下示例。

$$\begin{aligned} \mathsf{unify}(\forall\alpha^{\mathsf{T}}.\,\alpha; \forall\alpha^{\mathsf{T}}.\,?\mathrm{X}) &= \mathsf{unify}([\alpha \mapsto \beta]^*\alpha; [\alpha \mapsto \beta]^*?\mathrm{X}) \\ &= \mathsf{unify}(\beta; ?\mathrm{X}) \\ &= \langle [?\mathrm{X} \mapsto \beta]; \varnothing \rangle \end{aligned}$$

该算法返回一个空约束集，任何模型都满足该约束集，尤其是恒等替换。然而，如果我们将该替换应用于输入类型，会得到类型$\forall\alpha^{\mathsf{T}}.\,\alpha$和 $\forall\alpha^{\mathsf{T}}.\,\beta$，它们并不等价。对于效应多态类型，也可以构造出类似的反例： $\forall\alpha^{\mathsf{E}}.\,\tau_1 \to_\alpha \tau_2$ 和 $\forall\alpha^{\mathsf{E}}.\,\tau_1 \to_{?\mathrm{X}} \tau_2$。此外，即使假设合一变量采用避免捕获语义，这个问题依然存在。

合一算法不正确，并不直接意味着类型推断算法不正确，因为用户无法直接编写含有合一变量的类型。不过，正如我们将在下一节看到的，类型推断算法可以在内部构造出这样的类型。

推断$(\Gamma; x) = \Gamma(x); \iota; \varnothing; \varnothing$ 推断$(\Gamma; \lambda x\colon \tau.\, e)$ =检查良构性$(\Gamma; \tau; \mathrm{T})$; 令 $\tau'; \varepsilon; \theta; \Omega$ =推断$(\Gamma, x\colon \tau; e)$ 于 $\tau \to_\varepsilon \tau'; \iota; \theta; \Omega$ 推断$(\Gamma; \lambda x.\, e)$ =新鲜生成 ?X 于 令 $\tau; \varepsilon; \theta; \Omega$ =推断$(\Gamma, x\colon ?\mathrm{X}; e)$ 于 $\theta(?\mathrm{X}) \to_\varepsilon \tau; \iota; \theta; \Omega$ 推断$(\Gamma; e_1\ e_2)$ =令 $\tau_1; \varepsilon_1; \theta_1; \Omega_1$ =推断$(\Gamma; e_1)$ 于 令 $\tau_2; \varepsilon_2; \theta_2; \Omega_2$ =推断$(\theta_1^*\Gamma; e_2)$ 于 新鲜生成 $?\mathrm{X}_t, ?\mathrm{X}_e$ 于 令 $\theta_3; \Omega_3$ =统一$(\theta_2^*\tau_1; \tau_2 \to_{?\mathrm{X}_e} ?\mathrm{X}_t)$ 于 $\theta_3(?\mathrm{X}_t); \varepsilon_1 \cdot \varepsilon_2 \cdot ?\mathrm{X}_e; \theta_3\theta_2\theta_1; \Omega_1 \cup \Omega_2 \cup \Omega_3$

推断(Γ**;** $\Lambda\alpha^\kappa.\, e$**) =let** τ**;** ε**;** θ**;** Ω **=推断(**Γ**,** α**::** κ**;** e**) in let** $\Omega' = \Omega \uplus \{\varepsilon \equiv \iota\}$ **in let** $\{I_i\}_{i=1\ldots n}$ **=自由变量(**Ω'**) \ 自由变量(**$\theta^*\Gamma, \alpha$**::** κ**) in 新鲜生成** β **in 新鲜生成 ?X1, . . . , ?X**n **in** $\forall\alpha^\kappa.\, \tau$**;** ι**;** θ**; (**$[\alpha \mapsto \beta][I_i \mapsto ?\mathrm{X}_i]$**)** $^*\Omega' \cup \Omega'$ **推断(**Γ**;** e **[D]) =let** τ'**;** ε**;** θ**;** Ω **= 推断(**Γ**;** e**) in match** τ' **with** ~~$\forall\alpha^\kappa.\, \tau$~~ **=>检查良构性(**$\Gamma$**; D;** κ**);** $[\alpha \mapsto \mathbf{D}]^*\tau$**;** ε**;** θ**;** $[\alpha \mapsto \mathbf{D}]^*\Omega$ **| _ =>fail**

图5。类型推断算法。

4 推断算法

类型推断算法见图5。假设输入表达式不包含合一变量，且所有绑定类型变量均具有不同的名称。该算法返回一个元组，其中包括推断类型、效应、针对类型合一变量的替换，以及效应之间的约束集。[3]

变量、两种 λ 抽象以及函数应用的情形都很直接。我们仅指出，在带注释的 λ 抽象情形中，原始呈现省略了对类型良构性的检查。这一省略很可能是有意为之的，因为该算法假设所有绑定类型变量彼此不同，因此这一良构性检查可能会在程序 α重命名的早期阶段完成。

在类型抽象情况下，推断主体的类型和效应后，算法首先将约束 $\varepsilon \equiv \iota$ 添加到约束集中，以强制执行纯性限制。然后，算法确保所得约束集相对于变量 α是多态的。为此，它通过将所有约束的副本纳入其中来模拟多态函数的实例化，并将 α替换为一个新常量 β。此外，为确保多态性检查不会影响所传播的约束 Ω'，所有未出现在环境中的变量（无论是在定义域中，还是在分配给变量的类型中）都会被替换为新鲜合一变量。请注意，绑定变量 α 可能会通过约束 Ω'逃逸出其作用域；这很可能也是有意为之的，正如我们将在类型应用情况下看到的那样。

类型应用的情况很直接，不过有两点例外。首先，原始呈现中省略了对描述符的良构性检查。与带注释的λ情况类似，这一省略很可能是有意为之。其次，算法不仅在多态类型的主体中用描述符 D 替换绑定类型变量 α，还会在约束集中进行替换

[3]我们更改了结果中最后两个组成部分的顺序，以与合一算法保持一致。

。原始呈现没有解释此替换的目的，我们也不清楚其用意。不过，此替换很可能是有意为之，因为可靠性证明中再次进行了同样的替换。我们无法从证明中推断出进行此替换的原因，但这似乎与以下事实有关：在类型抽象情况下，绑定类型变量会通过约束逃逸出其作用域。然而，如果省去此替换，所给出的（部分）证明会更简单。此外，这种奇怪替换中的变量 α 来自对绑定符 $\forall \alpha^{\kappa}. \tau$ 的展开，这表明绑定变量的实际名称很重要，而且合一变量的捕获语义是有意为之的。

4.1 可靠性

据称，所提出的类型推断算法相对于声明式类型系统是可靠的。可靠性定理陈述如下。

定理 4.1（可靠性，[24]中的定理 3）。如果推断$(\Gamma;\ e) = \langle \tau;\ \varepsilon;\ \theta;\ \Omega \rangle$ 且 $m \models \Omega$ 对于某个模型 m，那么 $m^*\theta^*\Gamma \vdash e :\ m^*\tau \,/\, m^*\varepsilon$。

所给出的证明仅关注应用和类型应用情形。然而，我们发现，证明未涵盖的其他情形存在问题。下面，我们总结已发现的问题。其中一个问题导致了一个反例，表明所提出的定理并不成立。

合一错误。正如我们在第 3.3 节中所见，合一算法在处理多态类型时存在错误。事实证明，类似于该节所述反例的情况可能会出现在类型推断算法中。考虑以下程序。

$$\lambda f : (\forall \alpha^{\mathsf{T}}. \alpha \to_{\iota} \alpha) \to_{\iota} \tau. f\ (\Lambda \beta^{\mathsf{T}}. \lambda x. x)$$

对于这个程序，类型推断算法会触发类型的合一 $\forall \alpha^{\mathsf{T}}. \alpha \to_{\iota} \alpha$ 和$\forall \beta^{\mathsf{T}}.? \ \mathsf{X} \to_{\iota} ? \ \mathsf{X}$，这会导致与第 3.3 节中相同的错误行为。不过，在这种情况下，整个程序是可定型的，而且整个类型推断算法会返回正确的类型。若不扩展演算，仅凭这一问题，我们无法构造出针对可靠性定理的反例。

类型抽象情况下的逃逸变量。在类型抽象情况下，算法无法确保绑定变量 α 不会逃逸出其作用域。这使得类型推断算法不可靠。例如，考虑以下程序。

$$\lambda f. \Lambda \alpha^{\mathsf{T}}. \lambda x : \alpha. f\ x$$

这个程序在声明式类型系统中不可定型，但类型推断算法却接受了它，并执行以下步骤（为便于阅读，我们将与纯效应的合并简化处理）。

- 首先，它会创建一个新鲜合一变量 ?Z，表示 f 的类型。
- 接着，对于最内层的 λ 抽象，算法推断出类型$\alpha \to_{?\mathsf{X}} ?\mathsf{Y}$，并得到替换 $[?\mathsf{Z} \mapsto \alpha \to_{?\mathsf{X}} ?\mathsf{Y}]$，其中 ?X 和 ?Y 是在应用情形中新鲜生成的统一变量。
- 对于类型抽象，算法返回类型 $\forall \alpha_{\mathsf{T}}. \alpha \to_{?\mathsf{X}} ?\mathsf{Y}$、相同的替换以及两个平凡约束副本 $\iota \equiv \iota$。请注意，绑定变量 α 通过替换逃逸出了其作用域。
- 最后，算法返回类型 $(\alpha \to_{?\mathsf{X}} ?\mathsf{Y}) \to_{\iota} \forall \alpha_{\mathsf{T}}. \alpha \to_{?\mathsf{X}} ?\mathsf{Y}$，以及同一组可平凡满足的约束。

可以构造一个类似的反例，其中效应变量逃逸出其作用域：

$$\lambda f. \Lambda \alpha^{\mathsf{E}}. \lambda x : \tau \to_{\alpha} \tau. f\ x,$$

其中 τ 是某种基本类型。要修正算法以防止类型变量逃逸相当容易，但效应变量的逃逸似乎是算法设计中固有的问题：算法只收集效应之间的约束，直到类型推断即将结束时才对它们进行分析。

4.2 完备性

类型推断算法也被称为相对于声明式类型系统是完备的。完备性定理表述如下。

定理 4.2（完备性，[24]中的定理 4）。若 $m^*\theta^*\Gamma \vdash e : \tau / \varepsilon$，则 $推断(\Gamma; e) = \langle \tau'; \varepsilon'; \theta'; \Omega' \rangle$，其中存在某些 τ'、ε'、θ' 和 Ω'，使得存在一个模型 m' 和一个替换 θ'' 满足以下条件：

- $m'^*\theta''^*\theta'^*\Gamma(x) = m^*\theta^*\Gamma(x)$ 对每个 $x \in 自变量(e)$，
- $m' \models \Omega'$，
- $\tau \equiv m'^*\theta'^*\tau'$，
- $\varepsilon \equiv m'^*\varepsilon'$。

给出的证明只关注 λ 抽象和多态抽象这两种情况。由于多态抽象（$\Lambda\alpha^\kappa.\, e$）的情况涉及变量绑定和替换，我们来更详细地考察证明中的以下部分（我们已更改记号，使其与本文的表述保持一致）。

> “[...] 根据归纳假设，存在 τ_0; ε_0; θ_0; Ω_0、m_0 和 θ'_0，它们能验证环境 $m^*\theta^*\Gamma$ 中 e，$\alpha :: \kappa$ 的定理。[...] 为证明该定理，取：
>
> $$\tau' = \forall\alpha^\kappa.\, \tau_0 \qquad \varepsilon' = \iota \qquad \theta' = \theta_0$$
>
> $$\Omega' = [...] \qquad m' = m_0[\overline{?\mathrm{X}_i \mapsto m_1(I_i)}] \qquad \theta'' = \theta'_0$$
>
> [...]。除约束的验证外，该定理显然成立。 [...]”

让我们使用上述选择来验证定理的第三个条件（针对类型）。我们需要证明

$$\forall\alpha^\kappa.\, \tau \equiv m'^*\theta''^*\forall\alpha^\kappa.\, \tau_0.$$

根据归纳假设，我们有 $\tau \equiv m_0^*\theta_0^*\tau_0$，并且可以忽略 $\overline{[?\mathrm{X}_i \mapsto m_1(I_i)]}$ 的 m'，因为合一变量 $?\mathrm{X}_i$ 不会出现在 τ_0 中。因此，我们需要证明

$$\forall\alpha^\kappa.\, m_0^*\theta_0^*\tau_0 \equiv m_0^*\theta_0^*\forall\alpha^\kappa.\, \tau_0.$$

由于我们使用捕获替换，这个等价关系显然成立。然而，如果这里指的是避免捕获的替换，那么我们就必须证明变量 α 不会出现在替换 θ_0 中。对此并无保证，因此在这种情况下证明是不正确的。这也进一步表明，预期采用的是合一变量的捕获语义。

然而，我们发现完备性定理并不成立。问题出在函数应用和多态应用情形中，而这两种情形在所给出的证明中都被省略了。第一个问题相当容易修正，第二个问题则更为严重。我们将在下文讨论这两个问题。

具有多态结果的箭头类型。在函数应用情形中，算法将函数的类型与类型 $\tau_2 \to_{?\mathrm{X}_e} ?\mathrm{X}_t$ 合一，其中 τ_2 是参数的类型，而 $?\mathrm{X}_e$ 和 $?\mathrm{X}_t$ 是新鲜生成的合一变量。但类型合一变量只能表示单态类型，因此算法不允许应用具有多态结果类型的函数。声明式类型系统中没有类似的限制，因此类型推断算法并不完备。作为反例，请看下面的程序。

$$(\lambda x.\, \Lambda\alpha^{\mathsf{T}}.\, x)\ 42$$

该程序在声明式类型系统中是可类型化的，但在应用情形中，算法需要将 ?Y $\to_\iota \forall\alpha^{\mathsf{T}}.?\mathrm{Y}$ 与 Int $\to_{?\mathrm{X}_e} ?\mathrm{X}_t$ 合一，却会因此失败。

这个问题有一个直接的解决办法。与类型应用情况类似，我们可以对函数的类型进行模式匹配，并且只统一参数的类型。

实例化前进行替换。所介绍的算法假定合一变量具有捕获语义。这意味着，出现在类型量词下的合一变量可以实例化为包含该量词所绑定变量的类型。然而，在类型应用情况下，实例化多态类型 $\forall\alpha^{\kappa}.\tau$ 时，算法会将描述符替换为类型 τ 中的绑定变量 α，却忽略了这样一个事实：τ 中的合一变量可能表示包含 α 的类型。

考虑未标注的多态恒等函数 $\Lambda\alpha^{\mathsf{T}}.\lambda x.x$。对于这个函数，算法推断出的类型为 $\forall\alpha^{\mathsf{T}}.?\mathrm{X} \to_\iota ?\mathrm{X}$，其中 ?X 是一个新鲜生成的合一变量。注意，多态类型的主体并不直接包含 α，但 ?X 可能会实例化为包含 α 的类型。现在，考虑将这个函数应用于某个类型 τ。算法会用 τ 替换 α，但这种替换不会产生任何作用！此类应用的类型将被推断为 ?X $\to_\iota$?X。

基于这个问题构造反例并不简单，因为在所介绍的演算中，类型中包含合一变量的每个多态函数最多只能使用一次（被应用，或作为参数传递给另一个函数）。不过请注意，?X 表示一种单态类型，但多态类型可以实例化为任意类型描述符，甚至可以是多态类型。考虑以下程序。

$$(\Lambda\alpha^{\mathsf{T}}.\lambda x.x)\ [\forall\beta^{\mathsf{T}}.\beta \to_\iota \beta]\ (\Lambda\gamma^{\mathsf{T}}.\lambda y.y)$$

这个程序在声明式类型系统中是可类型化的，但类型推断算法尝试将合一变量实例化为多态类型时失败了。

如果我们用 let 绑定（let $x = e_1$ in e_2）扩展演算，并采用通常的类型规则，以及以下对类型推断算法的扩展，就可以更直接地构造一个反例。[4]

$$\begin{aligned}
&\mathsf{infer}(\Gamma; \mathtt{let}\ x\ =\ e_1\ \mathtt{in}\ e_2) = \\
&\quad \textbf{let}\ \langle \tau_1; \varepsilon_1; \theta_1; \Omega_1\rangle = \mathsf{infer}(\Gamma; e_1)\ \textbf{in} \\
&\quad \textbf{let}\ \langle \tau_2; \varepsilon_2; \theta_2; \Omega_2\rangle = \mathsf{infer}(\theta_1^*\Gamma, x:\tau_1; e_2)\ \textbf{in} \\
&\quad \langle \tau_2; \varepsilon_1 \cdot \varepsilon_2; \theta_2\theta_1; \Omega_1 \cup \Omega_2\rangle
\end{aligned}$$

现在，考虑以下程序。

$$\begin{aligned}
&\Lambda\alpha_1^{\mathsf{T}}.\Lambda\alpha_2^{\mathsf{T}}.\lambda x_1:\alpha_1.\lambda x_2:\alpha_2. \\
&\quad \mathtt{let}\ f\ =\ \Lambda\beta^{\mathsf{T}}.\lambda y.y\ \mathtt{in} \\
&\quad \mathtt{let}\ z\ =\ f\ [\alpha_1]\ x_1\ \mathtt{in} \\
&\quad f\ [\alpha_2]\ x_2
\end{aligned}$$

同样，这个程序在声明式类型系统中是可类型化的。然而，类型推断算法在执行以下步骤时失败了。

- 首先，它将类型 $\forall\beta_{\mathsf{T}}.?\mathrm{X} \to_\iota ?\mathrm{X}$ 赋给 f，其中 ?X 是一个新鲜合一变量。
- 类型应用 $f\ [\alpha_1]$ 会得到类型 ?X $\to_\iota$?X，与前一个示例相同。

[4]在原论文的结尾，作者讨论了一种更一般形式的 let 绑定，它允许某种形式的 let 多态。不过，就我们的目的而言，只需考虑其较简单的形式，并假设所有表达式都是扩展的。

- 将 $f\ [\alpha_1]$ 应用于 x_1 会使统一变量 ?X 实例化为类型 α_1。现在，f的类型是 $\forall\beta_T.\,\alpha_1 \to_\iota \alpha_1$。它不再依赖于 β。
- 最后，x_2 的类型与 α_1不匹配，因此无法将 $f\ [\alpha_2]$ 应用于 x_2。

可以基于效应多态性构造一个类似的反例，它会产生不可满足的效应约束，而不是类型不匹配。

5 结论

在存在高秩多态的情况下，类型推断算法很容易看起来完全正确，实则包含许多微妙的变量绑定错误。在纸笔证明中，这些问题也很容易被忽视，因为对变量和替换的操作往往被视为一种繁琐工作，而非推理中的重要部分。借助证明助手可以避免这类问题，但人们仍花了很长时间才开发出能够精确处理变量绑定、又不至于枯燥乏味的技术。自 [1, 6, 9, 18, 22, 41–44], 以来，人们开发了若干针对复杂演算的类型推断形式化方法，这些方法得以实现，正是因为能够精确跟踪变量作用域。值得一提的是，这种精确程度并非机械化证明所独有，传统的纸笔证明也可以做到 [25]。我们希望，随着证明助手日益普及，这种精确程度能够成为该研究领域的标准。

参考文献

[1] Patrycja Balik、Szymon Jędras 和 Piotr Polesiuk。2026年。Deciding not to Decide：高秩多态性存在时健全且完备的效应推断。已被 ESOP 2026 接收。[2] Andrej Bauer 和 Matija Pretnar。2015年。使用代数效应和处理器进行编程。《逻辑、代数与程序方法杂志》。 84, 1 2015 108 123 10 1016 2014 02 001 ( ), –。https://doi.org/ 。/j.jlamp. 。。
[3] Dariusz Biernacki、Maciej Piróg、Piotr Polesiuk 和 Filip Sieczkowski。 。抽象代数效应。ACM程序设计语言会议录。 3, 2019 6 1 6 28 10 1145 3290319 POPL（ ），：–：。doi：。/ Dariusz Biernacki、Maciej Piróg、Piotr Polesiuk 和 Filip Sieczkowski。2020年。白天是绑定符，夜晚是标签：通过词法作用域处理器实现效应实例。PACMPL 4, 2020 48 1 48 29 10 1145 3371116 POPL（ ），：–：。https://doi.org/ 。/ Lars Birkedal、Mads Tofte 和 Magnus Vejlstrup。 。从区域推断到冯·诺依曼机：通过区域表示推断。在《POPL'96论文集：第23届ACM SIGPLAN-SIGACT编程语言原理研讨会论文集》中，论文发表于1996年1月21日至24日，美国佛罗里达州圣彼得堡海滩，Hans-Juergen Boehm 和 Guy L. Steele Jr.（编）。ACM出版社， 171 183 10 1145 237721 237771 –。https://doi.org/ 。/ 。
[6] Roger Bosman、Georgios Karachalias 和 Tom Schrijvers。 。一个统一变量也不遗漏：HDM 系统的完全实例化类型推断。在第14届交互式定理证明国际会议（ITP 2023）论文集中，2023年7月31日至8月4日，波兰比亚韦斯托克（LIPIcs，第268卷），Ada m Naumowicz 和 René Thiemann（编）。Schloss Dagstuhl - Leibniz信息学中心，8:1–8:18。https://doi.org/10.4230/LIPIcs.ITP.2023.8[7] Didier Le Botlan 和 Didier Rémy。2003年。MLF：将 ML 提升到 F 系统的能力。在第八届ACM SIGPLAN函数式编程国际会议论文集（ICFP 2003）中，2003年8月25日至29日，瑞典乌普萨拉，Colin Runciman 和 Olin Shivers（编）。ACM， 27 38 10 1145 944705 944709 –。https://doi.org/ 。/ 。
[8] Jonathan I. Brachthäuser、Philipp Schuster 和 Klaus Ostermann。 。将效应视为能力：效应处理器与轻量级效应多态性。PACMPL 4, 2020 126 1 126 30 10 1145 3428194 OOPSLA（ ），：–：。https://doi.org/ 。/ Chen Cui、Shengyi Jiang 和 Bruno C. d. S. Oliveira。 。贪心隐式有界量化。ACM程序设计语言会议录。 7, 2 2023 2083 2111 10 1145 3622871 OOPSLA（ ），–。https://doi.org/ 。/ Paulo Emílio de Vilhena 和 François Pottier。 。效应处理器与动态标签的类型系统。在《编程语言与系统——第32届欧洲编程研讨会（ESOP 2023）》论文集中，该会议作为欧洲软件理论与实践联合会议（ETAPS 2023）的一部分于2023年4月22日至27日在法国巴黎举行（《计算机科学讲义》，第13990卷），Thomas Wies（编）。施普林格，225–252。https://doi.org/10.1007/978-3- 031-30044-8 9[11] Gilles Dowek、Thérèse Hardin 和 Claude Kirchner。2000年。通过显式替换实现高阶合一。《信息与计算》。 157, 1 2 2000 183 235 10 1006 1999 2837 - （ ），–。doi：。/INCO. 。
[12] Catherine Dubois 和 Valérie Ménissier-Morain。 。ML 类型推断工具的认证：Coq 中的 Damas-Milner。《自动推理杂志》23, 3–4（1999年），319–346。https://doi.org/10.1023/A:1006285817788

[13] Jana Dunfield 和 Neelakantan R. Krishnaswami。2013年。高秩多态的完备且易用的双向类型检查。在ACM SIGPLAN函数式编程国际会议（ICFP'13）论文集中，2013年9月25日至27日，美国马萨诸塞州波士顿，Greg Morrisett 和 Tarmo Uustalu（编）。ACM，429–442。https://doi.org/10.1145/2500365. 2500582[14] Jana Dunfield 和 Neelakantan R. Krishnaswami。2019年。含存在类型和索引类型的高秩多态的健全且完备的双向类型检查。ACM程序设计语言会议录3, POPL（2019年），9:1–9:28。https://doi.org/ 10 1145 3290322 。/ Martin Elsman。 。ReML 中的显式效应与效应约束。ACM程序设计语言会议录8，POPL（2024年），[15] 2024 2370 2394 10 1145 3632921 –。https://doi.org/ 。/ Frank Emrich 、 Sam Lindley 、 Jan Stolarek 、 James Cheney 和 Jonathan Coates。2020 年。FreezeML：一等多态的完备且易用的类型推断。在第41届ACM SIGPLAN编程语言设计与实现国际会议论文集（PLDI 2020）中，2020年6月15日至20日，英国伦敦，Alastair F. Donaldson 和 Emina Torlak（编）。ACM，[16] 423 437 10 1145 3385412 3386003 –。https://doi.org/ 。/ 。Manuel Fähndrich 和 Alexander Aiken。 。使用混合项约束和集合约束进行程序分析。在《静态分析》第四届国际研讨会（SAS' 97）论文集中，1997年9月8日至10日，法国巴黎（《计算机科学讲义》，第[17] 1997 *1302* 114 126 10 1007 0032737 卷），Pascal Van Hentenryck（编）。施普林格，–。https://doi.org/ 。/BFb Andong Fan、Han Xu 和 Ningning Xie。 。基于层级的实用类型推断。ACM程序设计语言会议录9，PLDI（[18] 2025 2025 2180 2203 10 1145 3729338 ），–。https://doi.org/ 。/ Simon Helsen 和 Peter Thiemann。 。ML 的多态特化。《ACM程序设计语言系统汇刊》26, 4（[19] 2004 2004 652 701 10 1145 1011508 1011510 ），–。https://doi.org/ 。/ 。Daniel Hillerström 和 Sam Lindley。 。借助行和处理器释放效应。在第一届类型驱动开发国际研讨会论文集（TyDe@ICFP 2016）中，2016年9月18日，日本奈良，James Chapman 和 Wouter Swierstra（编）。ACM，[20] 2016 15 27 10 1145 2976022 2976033 –。https://doi.org/ 。/ 。Gérard P. Huet。 。多年后的高阶合一。在《高阶逻辑定理证明》第15届国际会议（TPHOLs 2002）论文集中，2002年8月20日至23日，美国弗吉尼亚州汉普顿（《计算机科学讲义》，第2410卷），Victor Carreño、César A. Muñoz 和 Sofiène Tahar（编）。施普林格，3–12。doi:10.1007/3-540-45685- 6 2_[22] Shengyi Jiang、Chen Cui 和 Bruno C. d. S. Oliveira。2025年。结合交并类型的双向高秩多态。ACM程序设计语言会议录[21] 2002 30 9, 2025 2118 2148 10 1145 3704907 POPL（ ），–。https://doi.org/ 。/ Pierre Jouvelot。9月。个人通信。[24] Pierre Jouvelot 和 David K. Gifford。1991年。类型与效应的代数重构。在《第十八届ACM编程语言原理年度研讨会论文集》中，1991年1月21日至23日，美国佛罗里达州奥兰多，303 310 10 1145 99583 – David S. Wise（编）。ACM出版社，–。https://doi.org/ 90623 /[23] 。[25] 1996 Andrew J Kennedy。 。类型推断与等式理论。技术报告 LIX/RR/96/09。巴黎综合理工学院。[26] Daan Leijen。2014年。Koka：使用行多态效应类型进行编程。在第五届数学结构化函数式编程研讨会论文集（MSFP@ETAPS 2014）中，2014年4月12日，法国格勒诺布尔（EPTCS，第[27] *153* 100 126 10 4204 153 8 卷），Paul B. Levy 和 Neel Krishnaswami（编）。–。https://doi.org/ 。/EPTCS. 。Magnus Madsen。 。Flix编程语言的原理。在ACM编程与软件新思想、新范式与反思国际研讨会论文集（Onward! 2022）中，2022年12月8日至10日，新西兰奥克兰，Christophe Scholliers 和 Jeremy Singer（编）。ACM，112–127。https://doi.org/ 2022 10 1145 3563835 3567661 。/ 。Wolfgang Naraschewski 和 Tobias Nipkow。1999年。经验证的类型推断：Isabelle/HOL 中的 W算法。《自动推理杂志》[28] 23, 3 4 1999 299 318 10 1023 1006277616879 -（ ），–。https://doi.org/ 。/A: Flemming Nielson 和 Hanne Riis Nielson。 。并发 ML 多态行为的约束。在《计算逻辑中的约束》第一届国际会议（CCL'94）论文集中，1994年9月7日至9日，德国慕尼黑（《计算机科学讲义》，第845卷），Jean-Pierre Jouannaud（编）。施普林格，73–88。https://doi.org/10.1007/ BFb0016845[30] Hanne Riis Nielson、Torben Amtoft 和 Flemming Nielson。1998年。行为分析与安全条件：CML 案例研究。在《软件工程基础方法》第一届国际会议（FASE'98）论文集中，该会议作为欧洲软件理论与实践联合会议（ETAPS'98）的一部分于1998年3月28日至4月4日在葡萄牙里斯本举行（《计算机科学讲义》，第1382卷），Egidio Astesiano（编）。施普林格，255–269。https://doi.org/ 1994 10 1007 0053595 。/BFb Martin Odersky 和 Konstantin Läufer。1996年。让类型标注发挥作用。[29] [31] 在《POPL'96论文集：第23届ACM SIGPLAN-SIGACT编程语言原理研讨会论文集》中，论文发表于1996年1月21日至24日，美国佛罗里达州圣彼得堡海滩，Hans-Juergen Boehm 和 Guy L. Steele Jr.（编）。ACM出版社，54–67。https://doi.org/10.1145/237721.237729

[32] Simon Peyton Jones、Dimitrios Vytiniotis、Stephanie Weirich 和 Mark Shields。2007年。任意阶类型的实用类型推断。JFP 17, 1 2007 1 82 10 1017 0956796806006034 ( ), –。https://doi.org/。/S Gordon D. [33] Plotkin 和 Matija Pretnar。2013年。处理代数效应。Log. Methods Comput. Sci. 9, 4 (2013)。https://doi.org/ 2013 10 2168 9 4 23 2013 。/LMCS- (：) John Alan Robinson。1965年。基于归结原理的面向机器的逻辑。J. [34] ACM 12, 1 (1965)，23–41。doi:10.1145/321250.321253[35] Jean-Pierre Talpin 和 Pierre Jouvelot。1994年。类型与效应体系。Inf. Comput. 111, 2 (1994)，245–296。https://doi.org/ 1965 10 1006 1994 1046 。/inco。Mads Tofte 和 Lars [36] Birkedal。1998年。区域推断算法。ACM Trans. Program. Lang. Syst. 20, 4 (1998), 1998 724 767 10 1145 291891 291894 –。https://doi.org/。/。Mads Tofte 和 Jean-Pierre Talpin。1994年。使用区域栈 [37] 实现带类型的按值调用 λ 演算。载于《POPL'94会议记录：第21届ACM SIGPLAN-SIGACT编程语言原理研讨会》，美国俄勒冈州波特兰，1994年1月17日至21日，Hans-Juergen Boehm、Bernard Lang 和 Daniel M. Yellin（编）。ACM出版社， 1994 188 201 10 1145 174675 177855 –。https://doi.org/。/。Christian Urban、Andrew M. Pitts 和 [38] Murdoch Gabbay。2004年。名义合一。Theor. Comput. Sci. 323, 1-3 (2004)，473–497。doi:10.1016/J.TCS.2004.06.016[39] Nicolas Wu、Tom Schrijvers 和 Ralf Hinze。2014年。作用域中的效应处理器。载于《2014年ACM SIGPLAN Haskell研讨会论文集》，瑞典哥德堡，2014年9月4日至5日，Wouter Swierstra（编）。ACM，1–12。https://doi.org/ 10.1145/2633357.2633358 [40] Ningning Xie、Youyou Cong、Kazuki Ikemori 和 Daan Leijen。2022年。一等名称与效应处理器。ACM程序设计语言会议录 6, 2 2022 30 59 10 1145 3563289 OOPSLA ( ), –。https://doi.org/。/ Xu Xue 和 Bruno [41] C. d. S. Oliveira。2024年。上下文类型化。ACM程序设计语言会议录 8, ICFP (2024)，880–908。https://doi.org/ 2024 10 1145 3674655 。/ Jinxu Zhao 和 Bruno C. d. S. Oliveira。2022年。初等类型推断。载于《第36届欧洲面向对象 [42] 编程大会，ECOOP 2022，2022年6月6日至10日，德国柏林》（《计算机科学讲义》，第222卷），Karim Ali 和 Jan Vitek（编）。Schloss Dagstuhl - Leibniz-Zentrum für Informatik， 2 1 2 28 10 4230 2022 2 ： –: 。https://doi.org/。/ [43] LIPIcs.ECOOP.。Jinxu Zhao、Bruno C. d. S. Oliveira 和 Tom Schrijvers。2018年。多态子类型算法的形式化。载于《交互式定理证明——第9届国际会议，ITP 2018，作为联合逻辑会议的一部分举行，FloC 2018，英国牛津，2018年7月9日至12日，论文集》（《计算机科学讲义》，第10895卷），Jeremy Avigad 和 Assia Mahboubi（编）。施普林格， 2018 604 622 10 1007 978 3 319 94821 8_36 –。https://doi.org/。/ - - - - Jinxu Zhao、 [44] Bruno C. d. S. Oliveira 和 Tom Schrijvers。2019年。高阶多态类型推断的机械化形式化。ACM程序设计语言会议录 3, ICFP (2019)，112:1–112:29。https://doi.org/10.1145/3341716 2019